```
# Luis Antonio Ortega AndrÃ©s 13/01/2021
import numpy as np
import matplotlib.pyplot as plt
import stochastic_plots as stoch
import BM_simulators as BM
from scipy.integrate import quad
from scipy import stats
from typing import List, Set, Dict, Tuple, Optional


def simulate_continuous_time_Markov_Chain(
    transition_matrix: np.ndarray,
    lambda_rates: np.ndarray,
    state_0: int,
    M: int,
    t0: float,
    t1: float,
) -> Tuple[list, list]:

    """ Simulation of a continuous time Markov chain

    Parameters
    ----------
    transition_matrix :
        Square matrix of transition probabilities between states.
        Rows have to add up to 1.0.
    lambda_rates :
        Rates for each of the states
    state_0 :
       Initial state encoded as an integer n = 0, 1,...
    M :
        Number of trajectories simulated.
    t0 :
        Initial time in the simulation.
    t1 :
        Final time in the simulation.

    Returns
    -------

    arrival_times : list
       List of M sublists with the arrival times.
       Each sublist is a the sequence of arrival times in a trajectory
       The first of element of each sublist is t0.

    tajectories : list
        List of M sublists.
        Sublist m is trajectory compose of a sequence of states
        of length len(arrival_times[m]).
        All trajectories start from state_0.

    Example
    -------
    >>> import numpy as np
    >>> import matplotlib.pyplot as plt
    >>> import examen_ordinaria_PE_2020_2021 as pe
    >>> transition_matrix = [[  0,   1, 0],
    ...                      [  0,   0, 1],
    ...                      [1/2, 1/2, 0]]
    >>> lambda_rates = [2, 1, 3]
    >>> t0 = 0.0
    >>> t1 = 100.0
    >>> state_0 = 0
    # Simulate and plot a trajectory.
    >>> M = 1 # Number of simulations
    >>> N = 100 # Time steps per simulation
    >>> arrival_times_CTMC, trajectories_CTMC = (
    ...     pe.simulate_continuous_time_Markov_Chain(
    ...     transition_matrix, lambda_rates,
    ...     state_0, M, t0, t1))
    >>> fig, ax = plt.subplots(1, 1, figsize=(10,5), num=1)
    >>> ax.step(arrival_times_CTMC[0],
    ...         trajectories_CTMC[0],
    ...         where='post')
    >>> ax.set_ylabel('state')
    >>> ax.set_xlabel('time')
    >>> _ = ax.set_title('Simulation of a continuous-time Markov chain')
```

```
    """
    # Define return lists
    arrival_times = []
    trajectories = []

    # Compute each trayectorie independently
    for _ in range(M):
        # Initial values
        t = t0
        state = state_0
        auxiliar_a_times = [t0]
        auxiliar_trajectories = [state]

        # Simulation loop
        while(True):
            # Compute next arrival time
            t += stats.expon.rvs(scale = 1/lambda_rates[state], size = 1)[0]

            # Check arrival in inside time limit.
            if (t >= t1):
                break

            # Store arrival time
            auxiliar_a_times.append(t)

            # Compute next state using cumsum function. The next state
            # is the frist one in satisfy the cumsum condition.
            u = stats.uniform.rvs()
            state = np.where(np.cumsum(transition_matrix[state]) >= u)[0][0]

            # Store new state.
            auxiliar_trajectories.append(state)

        # Append simulation results.
        arrival_times.append(auxiliar_a_times)
        trajectories.append(auxiliar_trajectories)

    return arrival_times, trajectories


def price_EU_call(
    S0: float,
    K: float,
    r: float,
    sigma: float,
    T: float,
) -> float:
    """ Price EU call by numerical quadrature.

    Parameters
    ----------
    S0 :
        Intial market price of underlying.
    K :
        Strike price of the option.
    r :
        Risk-free interest rate (anualized).
    sigma :
        Volatility of the underlyi9ng (anualized).
    T :
        Lifetime of the optiom (in years).

    Returns
    -------
    price : float
        Market price of the option.

    Example
    -------
    >>> import numpy as np
    >>> import matplotlib.pyplot as plt
    >>> import examen_ordinaria_PE_2020_2021 as pe
    >>> S0 = 100.0
    >>> K = 90.0
    >>> r = 0.05
    >>> sigma = 0.3
    >>> T = 2.0
```

```
    >>> price_EU_call = pe.price_EU_call(S0, K, r, sigma, T)
    >>> print('Price = {:.4f}'.format(price_EU_call))
    Price = 26.2402
    """

    def integrand(z):
        """ Integrand of a European option. """
        S_T = S0 * np.exp((r - 0.5*sigma**2) * T + sigma * np.sqrt(T) * z)
        payoff = np.maximum(S_T - K, 0)
        return payoff * stats.norm.pdf(z)

    discount_factor = np.exp(- r * T)
    R = 10.0
    price_EU_call = discount_factor * quad(integrand, -R, R)[0]

    return price_EU_call


def price_EU_call_MC(
    S0: float,
    K: float,
    r:float,
    sigma: float,
    T: float,
    M: int,
    N: int
) -> Tuple[float, float]:

    """ Price EU call by numerical quadrature.

    Parameters
    ----------
    S0 :
        Intial market price of underlying.
    K :
        Strike price of the option.
    r :
        Risk-free interest rate (anualized).
    sigma :
        Volatility of the underlyi9ng (anualized).
    T :
        Lifetime of the optiom (in years).
    M :
        Number of simulated trajectories.
    N :
        Number of timesteps in the simulation.
    Returns
    -------
    price_MC : float
        Monte Carlo estimate of the price of the option
    stdev_MC : float
        Monte Carlo estimate of the standard devuation of price_MC

    Example
    -------
    >>> import numpy as np
    >>> import matplotlib.pyplot as plt
    >>> import examen_ordinaria_PE_2020_2021 as pe
    >>> S0 = 100.0
    >>> K = 90.0
    >>> r = 0.05
    >>> sigma = 0.3
    >>> T = 2.0
    >>> M = 1000000
    >>> N = 10
    >>> price_EU_call_MC, stdev_EU_call_MC = pe.price_EU_call_MC(S0, K, r, sigma, T,
M, N)
    >>> print('Price (MC)= {:.4f} ({:.4f})'.format(price_EU_call_MC,
stdev_EU_call_MC))
    """

    return price_MC, stdev_MC



def euler_maruyana(t0, x0, T, a, b, M, N):
    """Numerical integration of an SDE using the stochastic Euler scheme.
```

```
    x(t0) = x0
    dx(t) = a(t, x(t))*dt + b(t, x(t))*dW(t)    [ItÃ´ SDE]

Parameters
----------
t0 : float
    Initial time for the simulation
x0 : float
    Initial level of the process
T : float
    Length of the simulation interval [t0, t0+T]
a :
    Function a(t,x(t)) that characterizes the drift term
b :
    Function b(t,x(t)) that characterizes the diffusion term
M: int
    Number of trajectories in simulation
N: int
    Number of intervals for the simulation

Returns
-------
t: numpy.ndarray of shape (N+1,)
    Regular grid of discretization times in [t0, t0+T].
X: numpy.ndarray of shape (M, N+1)
    Simulation consisting of M trajectories.
    Each trajectory is a row vector composed of the values
    of the process at t.

Example
-------
>>> import matplotlib.pyplot as plt
>>> import sde_solvers as sde
>>> t0, S0, T, mu, sigma = 0, 100.0, 2.0, 0.3,  0.4
>>> M, N = 20, 1000
>>> def a(t, St): return mu*St
>>> def b(t, St): return sigma*St
>>> t, S = sde.euler_maruyana(t0, S0, T, a, b, M, N)
>>> _ = plt.plot(t,S.T)
>>> _= plt.xlabel('t')
>>> _=  plt.ylabel('S(t)')
>>> _= plt.title('Geometric BM (Euler scheme)')

"""
dT = T/N  # size of simulation step

# Initialize solution array
t = np.linspace(t0, t0 + T, N + 1)  # integration grid
X = np.zeros((M, N + 1))

# Initial condition
X[:, 0] = np.full(M, x0)

for n in range(N):
    dW = np.random.randn(M)
    X[:, n + 1] = (X[:, n] + a(t[n], X[:, n])*dT
                   + b(t[n], X[:, n])*np.sqrt(dT)*dW)

return t, X
```